順序列テスト基準に基づく並行処理プログラムのテスト充分性評価

伊東栄典[†] 川口豊[†,☆]
古川善吾[††] 牛島和夫[†]

並行処理プログラムの普及に伴い，その信頼性向上手法としてテストが重要になっている．一般にテストでプログラムの正しさを証明できない．このためテストの終了を判定するためのテスト充分性評価が必要になる．テスト充分性評価には，被覆率をテストの終了判定の指標として用いる方法が逐次処理プログラムで実用化されている．テスト実施前に抽出した測定事象と，テスト実施中に実行された事象との割合が被覆率である．テスト基準に基づき測定事象は定められる．逐次処理プログラムに比べ並行処理プログラムは動作が複雑である．そのため並行処理プログラムのテスト充分性評価に逐次処理プログラムで用いられているテスト充分性評価法を利用するだけでは十分ではない．並行処理プログラムに対する新たなテスト基準が必要である．本稿では並行処理プログラムに対するテスト基準として，プロセス間の同期や通信の実行順序に基づく「順序列テスト基準」を提案し，これを用いたテスト充分性評価法について検討する．順序列とは，並行に実行されるプロセス間での同期や通信を行う文の実行系列である．列の長さを変化させることにより，プログラムのさまざまな誤りに対応できる．長さ1の順序列テスト基準は文被覆テスト基準に「包含される」．長さ2の順序列テスト基準は通信誤りに対し「信頼できる」．順序列の長さをプロセス数+1にすれば単純なデッドロックに対し「信頼できる」．

Reliability of Testing based on Ordered Sequence Testing Criteria for Concurrent Programs

EISUKE ITOH,[†] YUTAKA KAWAGUTI,[†,☆] ZENGO FURUKAWA [††]
and KAZUO USHIJIMA [†]

Testing of concurrent programs is important for increasing reliability of the programs. Generally, it is impossible to prove correctness of programs through testing. Therefore, it is necessary to decide when the program testing is completed. The evaluation of testing reliability reveals us on condition for testing completion. Coverage represents the testing reliability objectively. Because the behavior of a concurrent program is more complex than that of a sequential program, it is not sufficient for concurrent programs to be evaluated of reliability with testing criteria for sequential programs. New testing criteria must be introduced for concurrent programs. This paper proposes new testing criteria, Ordered Sequence Criteria (**OSC**) for concurrent programs. **OSC** only pay attention to interprocess communication and synchronization. An $\mathbf{OSC}_k$ selects k-length sequences which consist of commands related to communication or synchronization. In testing, all sequences should be executed at least once. An $\mathbf{OSC}_k$ requires various levels of testing according to values of k. The $\mathbf{OSC}_{n+1}$ is reliable for a program which is correct or which includes simple deadlocks, where n represents the number of processes in the program.

1. はじめに

プログラムを作成する場合，開発費の約50%がテスト費用であると言われている[1]．このため，従来の逐次処理プログラムでは，テスト費用削減のためにテスト法の研究やテスト支援ツールの開発が進んでいる．現在，マルチプロセッサシステムやLocal Area Networkなどの普及に伴い並行処理プログラムが実際に

† 九州大学工学部情報工学科
Department of Computer Science and Communication Engineering, Kyushu University
†† 九州大学情報処理教育センター
Educational Center for Information Processing, Kyushu University
* 現在，沖電気工業株式会社
Presently with OKI Electronics Industry Corporation

用いられるようになってきた．このため並行処理プログラムの信頼性向上手法としてのテストが重要になってきている．

逐次処理プログラムに対するテスト法を，そのまま並行処理プログラムに適用するのでは，並行処理プログラムの特性を充分にテストできない．しかしながら，並行処理プログラムに対するテスト法は充分な研究がなされているとは言いがたく，またプログラム開発現場で実用に耐えうるツールも少ない．このため並行処理プログラムに対するテスト法の研究およびテスト支援ツールの開発が必要である．

並行処理プログラムのテストを難しくする原因の1つに並行処理プログラムの非決定的な動作がある．逐次処理プログラムでは，同じ入力に対する出力は一般に一定である．これに対し並行処理プログラムでは，プロセスの実行順序によって，同じ入力に対し出力が異なることがある．そのため並行処理プログラムをテストする際には，プログラムの入力（テストデータ）だけでなくプロセスの実行順序や命令の実行系列を考慮する必要がある．

テストでは，並行処理プログラムの正しさはテストを実施した際の入力および実行系列に対してのみ主張できる．並行処理プログラムの正しさをテストで証明するためには，プログラムの入力と実行系列との組合せをすべて実行しなければならない．一般にプログラムの入力値と実行系列との組の個数は膨大になる．そのためテストでプログラムの正当性を証明することは現実的に不可能である．

テストでプログラムの正当性を証明できないため，どこまでテストを行えば充分であるかをいかに評価するのかが問題となる．この評価をテスト充分性評価といい，テストに関する研究課題の1つとなっている．テスト充分性評価法の1つに，被覆率（coverage）を指標にする技法がある．被テストプログラムからテスト実施時に実現されるべき事象である測定事象を，テスト実施前にあらかじめ抽出する．テスト実施の際に実現された測定事象の割合が被覆率である．測定事象はテスト基準により定義される．

本稿では，並行処理プログラムに対するテスト基準として**順序列テスト基準**（Ordered Sequence Criteria：**OSC**）を提案し，その特性を定性的に分析する．これにより順序列テスト基準の利用可能性および理論的な限界を明確にできる．順序列テスト基準は，並行処理プログラムをインタリーブモデルで解釈した場合における命令の実行系列に基づいており，通信文や同期文の実行系列（順序列）を測定事象とするテスト基準である．

第2章は，テスト充分性評価法について考察する．第3章で順序列テスト基準背景および定義を述べる．第4章では，並行処理プログラムの例について順序列テスト基準およびテスト例を示す．第5章で，順序列テスト基準およびそれを用いたテスト法の特性について定性的に分析する．第7章ではまとめと今後の課題について述べる．

2. テスト充分性評価法

テストの充分性をプログラムの構造と実行時の測定とに基づいて評価するのがテスト充分性評価法である．逐次処理プログラムでテスト充分性評価に用いている被覆率（coverage）を並行処理プログラムのテスト充分性評価にも用いる．

被覆率を次のように定義する．まず，テストを行う前にプログラムの中から実行（被覆）されるべき測定事象を抽出する．この測定事象の集合の中で，テスト実施で実際に実行された測定事象の割合が被覆率である．被覆率 C の形式的な定義は次のとおりである．

【定義1】 被覆率 C

$$C=\frac{|W|}{|M|},$$

ここで M は被テストプログラム内の測定事象の集合を，W は実行された測定事象の集合を，$|\cdot|$ は集合の要素数を表す． □

被覆率を求めるためにはテストを実施する前に測定事象が抽出できなければならない．この測定事象はテスト基準により定義される．現在までに様々なテスト基準が提案されている[2)~9)]．

逐次処理プログラムでは，文被覆（all-statements）テスト基準や，枝被覆（all-branches）テスト基準，定義使用被覆（all-DU-paths）テスト基準などのテスト基準が実用化されている．文被覆テスト基準は C_0 テスト基準と，枝被覆テスト基準は C_1 テスト基準と，呼ばれることがある．また C_0 や C_1 を拡張した全路（all-paths）テスト基準がある．これはプログラム内に存在するすべての実行経路を測定事象とするテスト基準である．全路テスト基準は C_∞ テスト基準とも呼ばれる．

ここに挙げたテスト基準に基づいて算出される被覆率が100%に達しても，被テストプログラムの正しさを証明するものではない．しかしながら，これらの被覆率はテスト充分性を定量的に表す実用的な指標の1つとして用いられている．

逐次処理プログラムのテスト基準を用いて，並行処理プログラムのテスト充分性を評価するのは充分では

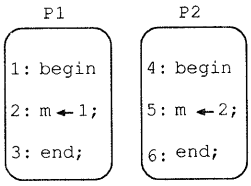


図1 非決定的実行の例
Fig. 1 An example of nondeterministic program.

ない．これは逐次処理プログラムのテスト法が並行処理プログラム特有の通信や同期，非決定性を考慮に入れていないためである．このことを簡単な例で示す．

図1のように，並行処理プログラム **P** が2つのプロセス P_1, P_2 を持つとする．このプログラムはプロセス P_1 がメモリ m に"1"を，P_2 が同じ m に"2"を書き込むものである．**P** を1度実行すれば P_1, P_2 内のそれぞれの全実行経路が被覆されたことになり，C_∞ テスト基準を満足する．逐次処理プログラムの場合，C_∞ テスト基準の被覆率が100%に達すればプログラム内のすべての誤りを検出する可能性がある．しかしながら，並行処理プログラム **P** の正しさは証明できない．2つのプロセスの代入文の実行順序が2つあるのでそのどちらもテストしなければならない．このように逐次処理プログラムのテスト基準だけで並行処理プログラムのテスト充分性を評価するのは妥当ではない．

並行処理プログラムのテスト充分性を評価するためには，並行処理プログラム特有の通信や同期，非決定性を考慮に入れたテスト基準が必要である．テスト基準は，定量的に評価が可能でかつ利用が容易でなければならない．

並行処理プログラムにおける被覆率を用いたテスト充分性評価についてはいくつかの提案が行われている[3),4),6),8),9)]．

Tai らは同期列（Syn-Sequence）を定義し，並行処理プログラムにおける非決定的な実行を同期列に従い決定的に再実行させる方法を提案している[8)]．並行処理プログラムを決定的に再実行させることにより誤りを再現することが可能になる．これによりプログラムのデバッグを容易にする．またデバッグだけでなく同期列をテストにおける測定事象と考える方法としても提案している[9)]．

古川らは Ada 並行処理プログラムを対象に，タスク間の通信および同期をランデブー通路として定義し，このランデブー通路をテストにおける測定事象とするテスト充分性評価法を提案している[3)]．また，共有変数のデータフローに着目した広域データフローテスト基準も提案している[4)]．これらは Ada 並行処理プログラムにのみ注目したテスト基準であるため一般の並行処理プログラムのテスト充分性を評価するものとは言いがたい．

Taylor らはプログラムのプロセスの状態の組合せ(並行状態)を節点とし，並行状態を変更する事象を枝とする並行状態グラフを定義した[6)]．テストを実施したときに実現された並行状態グラフの節点や枝の被覆率によってテスト充分性を評価する．しかしながら並行状態グラフには次の2つの問題点がある．

（1） 並行状態グラフの節点や枝の数がプロセス数に対して組合せ論的に増大する．

（2） 並行状態グラフを定義するためにはプロセスの数が静的に決まっていなければならない．

並行処理プログラムには，動的にプロセスが生成され，生成されたプロセス間で通信や同期を行いながら処理を進めるプログラムが多い．このためプロセス数が静的に決まっていなければならない並行状態グラフは実用的とは言いがたい．

本稿で提案する順序列テスト基準は，ソースコードに基づくテスト基準である．プロセス数がテストの実施前に決まっていなくとも，テストの実施前に順序列の長さを決めておけばソースコードからテストにおける測定事象が抽出できる．従って Taylor の並行状態グラフにおける問題点の(2)は回避できる．

3. 順序列テスト基準

3.1 並行処理プログラムの動作モデル

順序列テスト基準を定義する前に，並行処理プログラムの動作モデルについて考察する．ここではインタリーブモデルに基づいた動作モデルを考える[10)]．インタリーブモデルは，並行処理プログラムを単一のプロセッサで実行する場合を表すためのモデルである．

インタリーブモデル上では並行処理プログラムの動作を次のように考える．並行処理プログラム **P** は複数のプロセスから成るとする．プロセスは逐次的に命令を実行する．プロセッサは実行可能な命令を1つ選択して実行する．プロセス間の同期や通信に関する命令とそれらによる制約とを除き，どの命令を選出するかは任意である．プロセッサが実行する命令はプログラム内のプロセスに記述されている命令のみであり，その他の命令が実行されることはない．プログラム **P** 全体での命令の実行系列は，プロセッサの状態により変化する．

図1の並行処理プログラム **P** を実行する際，インタリーブモデル上で実現可能な実行系列の集合は，次のようになる．

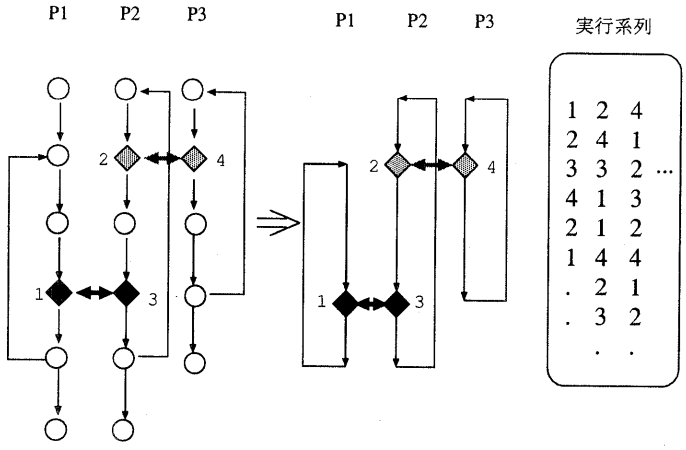


図2 同期・通信命令のみに注目した実行列の例
Fig. 2 An example of ordered sequences constructed by interprocess communication and synchronization.

$E(\mathbf{P})$={<123456>,
<124356>, <124536>, <124563>,
<142356>, <142536>, <142563>,
<145236>, <145263>, <145623>,
<412356>, <412536>, <412563>,
<415236>, <415263>, <415623>,
<451236>, <451263>, <451623>,
<456123>}.

実現可能な実行系列をすべて実行させればプログラム内のすべての誤りを検出できる可能性がある．しかしながら，一般の並行処理プログラムにおいて，実現可能な実行系列の数は組合せ論的に増大するし，無限ループがある場合には無限になる．実行可能な実行系列をすべて実行することは現実的に不可能である．実用的時間内にテストを終了するためにはより簡易なテストでなければならない．

並行処理プログラムの特徴は，複数のプロセスが同時に処理を行うために発生する，非決定性とプロセス間の通信や同期との2つである．そこで，並行処理プログラムの特徴を調べるテスト基準を考える．

インタリーブモデルでは，入力データが同じ場合，局所変数への代入文や参照文のプロセス間における実行順序がプログラム全体の実行結果を変化させることはない．通信や同期などの並行処理に関する文（通信同期文）の実行順序は結果を変化させる可能性がある．そこで図2のように，プロセス間の通信同期文だけに注目し，これらの命令を組み合わせた順序列を考える．この列をテストにおいて実行すべき事象とする．測定事象数を有限にするため，有限長の通信同期文の列を測定事象とする．

並行処理プログラムの特徴として，1つのソースコードから複数のプロセスを生成するプロセスの動的生成がある．図3は，ソースコードでは1つのプロセスであるけれども，実行時に複数のプロセスが生成され，互いに通信を行うプログラムである．このプログラムでは，通信同期文3が連続して実行される．同一の通信同期文を複数個含む列も測定事象に含める必要がある．

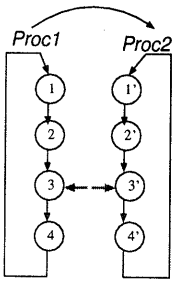


図3 プロセスの動的生成
Fig. 3 Dynamic Process Generation.

3.2 定　　義

以下に順序列テスト基準の形式的な定義を示す．

【定義2】 順序列テスト基準：$\mathbf{OSC}_k$，$k \geq 1$

$\mathbf{OSC}_k = \{< s_1, s_2, \cdots, s_k > | s_j \in Sync, 1 \leq j \leq k\}$,

ここで，s_j，$1 \leq j \leq k$，は通信同期文，$<\cdot>$ は順序列，$Sync$ はプログラム内のすべての通信同期文からなる集合（通信同期文集合）を表す．$\mathbf{OSC}_k$ はテストの際に実行されるべき測定事象の集合である． □

以後，通信同期文の順序列を単に順序列と表記する．

順序列テスト基準 $\mathbf{OSC}_k$ の k に具体的な値を与えた場合，順序列テスト基準の持つ意味は以下のとおりである．

$k=1$ の場合

$\mathbf{OSC}_1 = \{< s >\}$.

OSC_1 ではプログラム内に存在するすべての通信同期文それぞれがテストにおける測定事象である．OSC_1 は，プログラム内のすべての同期通信文を少なくとも1回実行することを要求するテスト基準である．C_0 テスト基準が満たされたならば，OSC_1 テスト基準も満たされる．

$k=2$ の場合

$OSC_2=\{<s_1, s_2>\}$.

OSC_2 では，プログラム内の通信同期文の長さ2の順序列がテストにおける測定事象である．OSC_2 テスト基準を満足すれば，プログラム内における2つのプロセス間の通信や同期を少なくとも1回実行する．Ada 並行処理プログラムにおけるランデブーテスト基準[3]や共有変数のデータフローテスト基準[4]は，この OSC_2 テスト基準の特別な場合である．

$k=3$ の場合

$OSC_3=\{<s_1, s_2, s_3>\}$.

OSC_3 では，プログラム内の通信同期文の長さ3の順序列がテストにおける測定事象である．OSC_3 テスト基準を満足するならば，プログラム内における3つのプロセス間の通信や同期を少なくとも1回実行する．

$k=\infty$ の場合

$OSC_\infty=\{<s_1, s_2, \cdots, s_k, \cdots>\}$.

ループが存在するプログラムでは，無限長の実行系列が存在する可能性がある．その場合，無限長の順序列を考えることが可能である．無限長の順序列をテストにおける測定事象にするのが OSC_∞ テスト基準である．OSC_∞ を満足するならば，プログラム実行時に起こりうる通信同期文の実行形列は少なくとも1回実行される．しかしながら，無限長の順序列が測定事象であるため，OSC_∞ テスト基準を満足することは現実的には不可能である．

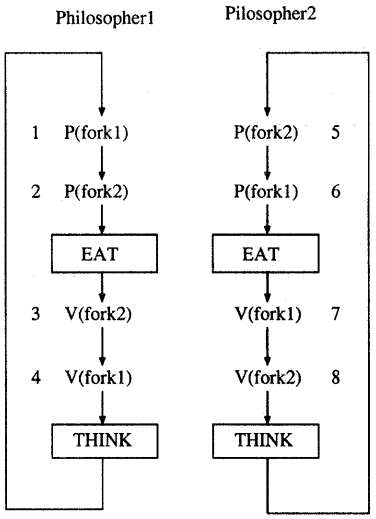


図4 哲学者2人の食事問題の制御フローグラフ
Fig. 4 A control flow graph of two dining philosophers.

4. 例 題

順序列テスト基準を具体的なプログラムに適用する．例題として「哲学者2人の食事問題」を用いる．制御フローグラフを図4に示す．同期の機構としてセマフォを用いた．

この例では，プログラムにおけるセマフォのP命令とV命令とが通信同期文集合 *Sync* の要素となる．まず，P，V命令に識別用の番号1～8を付ける（図4参照）．哲学者2人の食事問題における順序列テスト基準の測定事象は，図4に示した通信同期文の番号を用いると以下のようになる．

1) OSC_1

$= \{<1>, <2>, <3>, <4>, <5>, <6>, <7>, <8>\}$.

測定事象の数は $|Sync|=8$ 個である．

2) OSC_2

$$= \{<1,1>, <1,2>, <1,3>, \cdots <1,7>, <1,8>,$$
$$<2,1>, <2,2>, \quad \cdots \quad <2,8>,$$
$$\vdots \qquad\qquad \ddots \qquad \vdots$$
$$<8,1>, <8,2>, <8,3>, \cdots \quad <8,8>\}$$

測定事象の数は $|Sync|^2=8^2=64$ 個である．

3) OSC_3

$$= \{<1,1,1>, <1,1,2>, \cdots <1,8,7>, <1,8,8>,$$
$$<2,1,1>, <2,1,2>, \ddots \quad <2,8,8>,$$
$$\vdots \qquad\qquad \ddots \qquad \vdots$$
$$<8,1,1>, <8,1,2>, \cdots \quad <8,8,8>\}$$

測定事象の数は $|Sync|^3=8^3=512$ 個である．

順序列の長さをkとすると，順序列テスト基準$\mathbf{OSC}_k$における測定事象の数は$|Sync|^k$個になる．測定事象の数がkのべき乗で増加するため，kが大きな順序列テスト基準は実用的ではない可能性がある．そのために，テストに許される時間に応じてkの値を選択する必要がある．また，測定事象数の削減ついて5.3節で議論する．

上記の$k=3$の場合の測定事象，<8, 1, 2>という列は，哲学者の食事問題ではデッドロックに陥る列である．kの値を変更することによって順序列テスト基準$\mathbf{OSC}_k$において発見できる誤りの種類が変化する．誤りに対する順序列テスト基準の信頼性については5.4節で述べる．

5. 議　　論

前章まで並行処理プログラムの新たなテスト基準として，プロセス間の同期通信文の順序列に着目した順序列テスト基準を定義した．この章では順序列テスト基準とその他のテスト基準との関係，順序列テスト基準の測定可能性，測定事象数，信頼性について議論する．

5.1 テスト基準間の関係

順序列テスト基準$\mathbf{OSC}_k$, $k\geq 1$, と逐次処理プログラムのテスト基準C_i, $i\geq 0$, とについて，テスト基準の間の関係を検討する．

まずテスト基準間の関係として以下の「包含関係」を定義する．

【定義3】　包含関係

テスト基準Cr_1を満足すれば，必ずテスト基準Cr_2も満足する場合，Cr_1がCr_2を「**包含する**」といい，これを$Cr_1 \supset Cr_2$で表す．　□

順序列テスト基準，路解析に基づく逐次処理プログラムのテスト基準において，それぞれ以下の包含関係が成立することは自明である．

（1）　$C_{i+1} \supset C_i$, $i\geq 0$

（2）　$\mathbf{OSC}_{k+1} \supset \mathbf{OSC}_k$, $k\geq 1$

また，C_0テスト基準は，プログラム内のすべての実行文を少なくとも1回実行するというテスト基準である．プログラム内のすべての文が少なくとも1回実行されれば，プログラム内のすべての通信同期文が少なくとも1回実行される．ゆえに以下の包含関係が成立する．

（3）　$C_0 \supset \mathbf{OSC}_1$

$\mathbf{OSC}_1$はプログラム内のすべての通信同期文を少なくとも1回実行したという以上の意味を持たない．しかしながら，通信や同期を行う通信同期文の組が1組しかない場合や，通信や同期の仕様が決定しているプログラムでは，$\mathbf{OSC}_1$でテスト充分性を評価することができる．しかしながら，$\mathbf{OSC}_k$, $k\geq 2$, テスト基準は並行に実行されるプロセス間での通信同期文の組合せを含んでいるので，C_i, $i\geq 1$, との包含関係はない．

プログラム$\mathbf{P}$全体での命令の実行系列は，$\mathbf{P}$を構成するプロセスP_iにおける命令の実行順序を保存している．そのために，実行系列の集合は，各プロセスにおける順序を保存した命令列をシャッフル[11]した列の系列である．この系列すべてをテストにおける測定事象とし，被覆率を100%にすることを要求するテスト基準を考える．これを全実行系列テスト基準と呼びCC_∞と表すことにする．

【定義4】　全実行系列テスト基準：CC_∞

CC_∞＝｛すべての実現可能な実行系列｝．　□

全実行系列テスト基準CC_∞と順序列テスト基準$\mathbf{OSC}_k$, $k\geq 1$, 逐次処理プログラムの路解析のテスト基準C_i, $i\geq 0$, の間に以下の包含関係が成立することは明らかである．

（4）　$CC_\infty \supset \mathbf{OSC}_\infty$

（5）　$CC_\infty \supset C_\infty$

以上の包含関係をまとめたものを**図5**に示す．ただし，矢印は包含関係を表し，矢の根本のテスト基準が矢の先のテスト基準を包含することを意味する．

5.2 測定可能性

テスト基準を用いてテスト充分性を評価するためには，被テストプログラムの実行時に被覆率を測定できなければならない．順序列テスト基準における被覆率測定の可能性について検討する[12),13)]．

被覆率を用いたテスト充分性の評価は，逐次処理，並行処理に関わらず，次の手順で行われる．

（1）　測定事象の抽出

（2）　実行された測定事象の記録

（3）　被覆率の算出

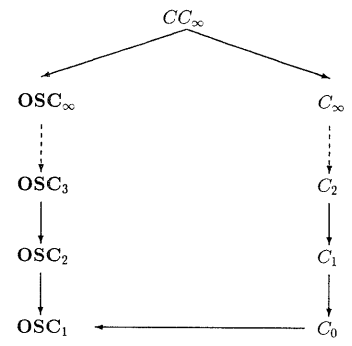


図5　テスト基準の間の包含関係
Fig. 5　Subsumption relationship of testing criteria.

それぞれの段階について検討する．

（1） 測定事象の抽出

順序列テスト基準はプロセス間の通信同期文からなる順序列を測定事象にしたテスト基準である．プログラムから通信同期文を取り出し，取り出した通信同期文の順序列を作成して定義1に示した測定事象の集合 M とする[13]．

（2） 実行された測定事象の記録

実行された測定事象の記録にはいくつかの方法がある．例えば実行時に実行状況の記録（トレース）を保存し，実行終了後にトレースから測定事象を抽出する方法がある．しかしながら，この方法は，トレースの量が膨大になるために，テスト充分性評価に用いるのには適していない．そこで，ソースコードに挿入した探針（Probe）によって実行した測定事象を記録する方法を用いる．探針とは被テストプログラムに挿入される測定用のプログラム片である．

順序列テスト基準では，プロセス間の通信同期文の順序列が測定事象である．そこで，被覆率を測定する探針の機能としては，並行に処理されるはずの同期通信文を何らかの形で逐次的に実行させることが必要である．逐次的に実行された通信同期文を順序列の長さ−1だけ保存すれば，測定事象である順序列が実行されたことを知ることが可能になる．

探針としては，監視用プログラム（モニタ）が考えられる．モニタは本来デバッグのために使用されてきた[8),14),15)]．プログラムの実行列を記録し，その実行列に従いプログラムを再実行させれば，並行処理プログラムを決定的に実行できるため，誤りが再現できデバッグが容易になる．

モニタを被覆率測定の探針として利用する手順は以下のとおりである．

（i） 被テストプログラムにモニタとなるプロセスを生成させるプログラム片を追加する．

（ii） 被テストプログラムの通信同期文に，モニタプロセスと通信をするための通信同期文を追加する．

（iii） 各プロセスでは通信同期文の実行前に，新たに追加された通信同期文を通じてモニタに自分のプロセス番号および次に実行される通信同期文の文番号を知らせ実行許可を求める．

（iv） モニタは実行許可要求を記録する．

（v） モニタは実行要求を出した通信同期文の実行許可を出す．

モニタは被覆率を増大させるために用いることも可能である．実行させたい順序列をモニタにあらかじめ知らせ，その順序列が実行されるようにモニタが各プロセスの実行順序を調節すれば良い．

モニタ挿入後のプログラムでは，並行に処理されるはずの通信同期文を逐次的に処理する．しかしながら，探針挿入後のプログラムの動作は元のプログラムの動作の1つである．なぜなら探針を挿入しても元のプログラムの実行系列における半順序関係を損なうものではないからである[16)]．そのため探針挿入後のプログラムで発生する誤りは，探針挿入前の被テストプログラムでも発生する．

（3） 被覆率の算出

最後に被覆率の算出を行う．一般にプログラムを一度実行しただけでは被覆率が100%になることはない．そのために，プログラムの実行を繰り返し，実行された測定事象を累積して被覆率を計算する必要がある．

プログラムの実行終了後にモニタが記録している実行した順序列を実行済み測定事象集合 M に加える．順序列テスト基準の被覆率 C を定義1の計算式で計算する．

5.3 測定事象数

長さ k の順序列テスト基準では，m^k 個（$m=|Sync|$）の組合せを測定事象とする．測定事象の数は順序列の長さのべき乗で増加するので，長い組合せを用いたテスト基準は実際的ではない可能性がある．そのために，測定事象を減らす必要がある．

測定事象の削減方法として考えられるのは，第1に，プログラムの意味を考慮することにより測定事象を削除することである．第2に，仕様に反する組合せを削除することである．しかしながらどちらの方法も採用しない．それぞれについて理由を述べる．

第1に，プログラムの意味を考慮することにより測定事象を削除することについて考える．3章で述べた<1, 1, 1>のような同一の通信同期文が並ぶ測定事象は，プログラムの意味から実現不可能と考えられる．しかし同一の通信同期文の列も実現可能な場合がある．

例えば，Adaのタスク型やUNIX上におけるCプログラムで用いられるシステムコールfork文などのように[17)]，同じソースコードに対し複数のプロセスが生成される場合がある．このような場合<1, 1, 1>といった同一の通信同期文の列がプログラムの実行系列に現れることがある．

次に生産者消費者問題を例に考える．バッファがすべて使用済みで空領域が無い場合，バッファに空領域ができるまで生産者プロセスは何度も通信を試みる．この場合にも，同一の通信同期文の列が実行系列に現れる．

第2に，仕様に反する系列を測定事象から削除することについて考える．被テストプログラムを計算機上で実際に実行するテストでは，仕様に反する事象あるいは仕様に記述されていない事象を発見することが目的である．仕様に反するという理由で測定事象から削除するのは主客転倒であり，この方法は採用できない．

つまりソースコードからだけでは，同一命令の組合せや仕様に反する組合せが実行系列に現れるか否かを判定できない．プログラムの動作を詳細に解析し，本当に実行できない測定事象を削除する，などの測定事象の削減法を今後検討する必要がある．

5.4 信 頼 性

テスト法が「**信頼できる**」とは，プログラムに誤りがあるならば，テスト法に従って作成したテストデータによって誤りを必ず発見できることをいう[5]．したがって，信頼できるテスト法により作成したテスト集合（テストデータ）に対してテストの実施結果がすべて正当であるならば，プログラムは正当である．任意のプログラムに対して信頼できるテスト法は「徹底テスト（exhaustive testing）」のみである．しかしながら徹底テストは一般に実現不可能であるため，現実に用いられているテスト法は部分的にしか信頼できない．そこで，特定の誤りを含んだプログラムに対してテスト法が信頼できるか否かを検討する必要がある．

5.1節の定義に従って，テスト基準 Cr_1 と Cr_2 に包含関係 $Cr_1 \supset Cr_2$ があると仮定する．ここでテスト基準 Cr_2 がある種類の誤りに対し信頼できるならば，Cr_2 を包含するテスト基準 Cr_1 も同じ種類の誤りに対し信頼できる．

並行処理プログラムの誤りは，誤りの発生原因により次の3種類に分類することが考えられる[10),18)]．

（i） プロセス内誤り……逐次処理プログラムにおいて考えられる誤り．

（ii） 通信誤り……並行に実行されるプロセス間でのデータの受け渡し時に発生する誤り．

（iii） 同期誤り……次の3種類がある．

（a） 安全性の破壊……相互排除の失敗によるデータの一貫性の喪失．

（b） 生存性の破壊……プログラム中のプロセスが実行不可能になる．デッドロックとも呼ばれる．

（c） 公平性の破壊……ある特定のプロセスだけが実行を不当に待たされる．ライブロックとも呼ばれる．

順序列テスト基準は，プロセス間の通信や同期に着目したテスト基準である．すなわち，各プロセスの中で発生するプロセス内誤りに対しては何ら考慮されていない．以下に通信誤りと生存性の破壊（デッドロック）との，2種類の誤りに対する順序列テスト基準の信頼性を考察する．

5.4.1 通信誤りに対する信頼性評価

通信誤りに対する順序列テスト基準の信頼性を検討する．通信誤りは次の2つに分類できる[3)]．

【定義5】 完全通信誤り

プロセス間における通信の際，常に誤ったデータを受け渡す． □

【定義6】 部分通信誤り

プロセス間における通信の際，誤ったデータを受け渡す場合がある． □

プロセス間の通信を対

$[s_1(\sigma), s_2(\sigma)]$

で表現する．ここで，σ は通信の際の識別子である．この通信が起こる場合の通信同期文の実行順序は $<s_1, s_2>$, $<s_2, s_1>$ のいずれかである．2つの列は長さ2の順序列テスト基準 $\mathbf{OSC}_2$ の測定事象に含まれている．

テスト実施後に $\mathbf{OSC}_2$ テスト基準を満足するならば，プログラム内の任意の2つのプロセス間のすべての通信が少なくとも1回は実行されたことになる．ゆえに，$\mathbf{OSC}_2$ テスト基準を満足するテスト法は，完全通信誤りに対し信頼できるテスト法である．

しかしながら，ある通信が部分通信誤りを含む場合，テスト実施においてその通信が一度しか実行されず，その際誤りが顕在しない場合がある．そのため $\mathbf{OSC}_2$ テスト基準は部分通信誤りに対しては信頼できない．

5.4.2 同期誤りに対する信頼性評価

同期誤りの中で，生存性の破壊であるデッドロックに対する順序列テスト基準の信頼性を検討する．デッドロックを定義する．

【定義7】 デッドロック：

プログラム内の少なくとも1つのプロセスが実行可能になることができない場合，これをデッドロックと呼ぶ． □

デッドロックを，その発生要因に基づき次のようにクラス分けする．

【定義8】 クラス0

プログラム内のある1つの通信同期文を実行すれば，必ず発生するデッドロック． □

【定義9】 クラス1

変数の値には関わらずに，プロセス間の通信同期文が，ある特定の順序列に従った場合にのみ発生するデッドロック． □

【定義 10】 クラス 2

通信同期文の順序列に関係なく，プログラム内の変数がある特定の値を持つ場合に発生するデッドロック． □

【定義 11】 クラス 3

プログラム内の変数がある特定の値を持ち，かつ通信同期文がある特定の順序列を持つ場合にのみ発生するデッドロック． □

それぞれのクラスのデッドロックに対する順序列テスト基準の信頼性は以下のとおりである．

[定理 1]

OSC_1 テスト基準はクラス 0 のデッドロックに対し信頼できる．

[証明]

クラス 0 のデッドロックでは，プログラム内のある 1 つの通信同期文を少なくとも 1 回実行すれば，必ずデッドロックが発生する．**OSC_1** は，プログラム内のすべての通信同期文の少なくとも 1 回の実行を要求する．もし **OSC_1** テスト基準を満足するならば，すべての通信同期文は少なくとも 1 回は実行されている．

ゆえに，**OSC_1** テスト基準はクラス 0 のデッドロックに対し信頼できる． □

[定理 2]

並行に実行されるプロセス数が n の場合，$n+1$ 長の順序列テスト基準を満足するテスト法は，通信同期文により発生するクラス 0 および 1 のデッドロックに対し信頼できる．

[証明]

クラス 1 のデッドロックは変数の値には関係せず，通信同期文の実行順序により発生する．変数の値に依存しないために，プロセス間の相互依存関係が循環する場合にのみデッドロックに陥る．プログラム中の並行に動作するプロセス数が n 個であるとする．プロセス間の相互依存関係の長さは高々 $n+1$ である．つまりプログラムにクラス 1 のデッドロックが存在する場合，長さ $n+1$ の順序列をすべて実行すれば，必ずデッドロックが発生する．

ゆえに，並行に実行されるプロセス数が n の場合，$n+1$ 長の順序列テスト基準を満足するテスト法は，通信同期文により発生するクラス 0 および 1 のデッドロックに対し信頼できる． □

[定理 3]

順序列テスト基準は，クラス 2 およびクラス 3 のデッドロックに対し信頼できない．

[証明]

反例を示す．

1: begin	5: begin
2: disk を x kbyte 確保	6: disk を y kbyte 確保
3: プロセス内の処理	7: プロセス内の処理
4: end	8: end

図 6 順序列テスト基準で発見不可能なデッドロックの例
Fig. 6 A program which can not detect deadlock with **OSC**.

例えば，**図 6** に示すような，変数に起因するデッドロックの場合，順序列テスト基準ではいかに列の長さを長くしても，このデッドロックを必ず発見するとはいえない． □

クラス 2 のデッドロックは，通信同期文の実行順序には関わらず，変数の値に起因するデッドロックである．順序列テスト基準は通信同期文にのみ着目したテスト基準であるため，プロセス内での変数の値の変化を何ら考慮していない．ゆえに順序列テスト基準はクラス 2 のデッドロックに対して信頼できない．

クラス 3 のデッドロックを発生する可能性のあるプログラムとしては，例えば複数のプロセスが資源を獲得しつつ処理を行い，しかもその資源の必要量が動的に変化するプログラムが考えられる．クラス 3 のデッドロックもクラス 2 と同様に変数の値に起因するデッドロックであるため，順序列テスト基準はクラス 3 のデッドロックに対して信頼できない．

6. おわりに

並行処理プログラムに対するテスト充分性評価法の 1 つとして，プロセス間の通信や同期に着目した順序列テスト基準を提案した．その順序列テスト基準の測定可能性，包含関係，信頼性について議論した．包含関係の議論で，従来のテスト基準と順序列テスト基準との関係を示した．信頼性の議論の中で，順序列テスト基準が信頼できるプログラムの限界を示した．順序列テスト基準は，正しいプログラム，完全通信誤りを含むプログラム，クラス 0 およびクラス 1 のデッドロックを含むプログラムに対して信頼できる．

今後の課題として，実際の開発現場における順序列テスト基準の有効性の評価がある．テストは実際の開発現場で使用されることを目的とする手法であるため，開発現場で有効でなければならない．そこで定性的な信頼性の検討だけでなく，実際のプログラム開発に適用して有効性を実証する必要がある．

本稿の考察に基づいて，順序列テスト基準の被覆率を用いたテスト充分性評価ツールの試作を現在進めている[13]．ツールは基本的な長さ 2 の順序列テスト基準の被覆率を測定する．ツールの対象は UNIX SYSTEM V および UNIX 4.3 BSD 上で実行可能な C 並

行処理プログラムである．C言語やUNIXは広く用いられているため，ツールは実際のプログラム開発における実用的な利用が期待できる．

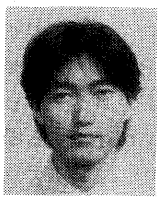
既に，小さいものではあるが，実際に使用されている並行処理プログラムにツールの適用を始めている．テスト基準およびツールの評価結果については別の機会に報告したい．今後は多くの並行処理プログラムにツールを適用し，順序列テスト基準の有効性を実証する予定である．

謝辞 並行処理プログラムのテスト/デバッグ/検証法について議論していただいた奈良先端科学技術大学院大学の荒木啓二郎教授および九州大学工学部情報工学科の程京徳助教授に感謝いたします．

参考文献

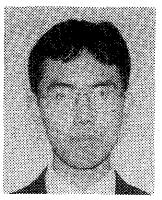
1) 玉井哲雄, 三嶋良武, 松田茂広：ソフトウェアのテスト技法, 共立出版 (1988).
2) Clarke, L. A., Podgurski, A., Richardson, D. J. and Zeil, S. J.: A Formal Evaluation of Data Flow Path Selection Criteria, *IEEE Trans. Softw. Eng.*, Vol. 15, No. 11, pp. 1318-1332 (1989).
3) 古川善吾, 牛島和夫：ランデブー通路を用いたAda並行処理プログラムのテスト十分性評価, 電子情報通信学会論文誌, Vol. J 75-D-I, No. 5, pp. 288-297 (1992).
4) 古川善吾, 有村耕治, 牛島和夫：並行処理プログラムにおける共有変数のデータフローテスト基準, 情報処理学会論文誌, Vol. 33, No. 11, pp. 1394-1401 (1992).
5) Howden, W. E.: Reliability of the Path Analysis Testing Strategy, *IEEE Trans. Softw. Eng.*, Vol. SE-3, No. 4, pp. 226-278 (1976).
6) Taylor, R. N., Levine, D. L. and Kelly, C. D.: Structural Testing of Concurrent Programs, *IEEE Trans. Softw. Eng.*, Vol. 18, No. 3, pp. 206-215 (1992).
7) Weyuker, E. J.: The Complexity of Data Flow Criteria for Test Data Selection, *Information Processing Letters*, Vol. 19, pp. 103-109 (1984).
8) Tai, K. C., Carver, R. H. and Obaid, E. E.: Debugging Concurrent Ada Programs by Deterministic Execution, *IEEE Trans. Softw. Eng.*, Vol. 17, No. 1, pp. 45-63 (1992).
9) Hwang G. H., Tai, K. C. and Huang, T.: Reachability Testing: An Approach to Testing Concurrent Software, *Proc. of APSEC94*, pp. 246-255 (1994).
10) Ben-Ari, M.: *Principles of Concurrent Programming*, Prentice Hall International, Inc. (1982), 渡辺榮一(訳)：並行プログラミングの原理, 啓学出版 (1986).
11) 荒木俊郎, 香西省治, 都倉信樹：シャッフル文法, 電子情報通信学会論文誌, Vol. J 66-D, No. 8, pp. 170-177 (1983).
12) Itoh, E., Kawaguti, Y., Furukawa, Z. and Ushijima, K.: Ordered Sequence Testing Criteria for Concurrnet Programs and The Support Tool, *Proc. of APSEC94*, pp. 236-245 (1994).
13) 川口 豊, 伊東栄典, 古川善吾, 牛島和夫：順序列テスト基準を用いたテスト充分性評価システムの試作, 情報処理学会研究会報告, 96-SE-14, pp. 107-114 (1994).
14) 程 京徳, 荒木啓二郎, 牛島和夫：Ada並列プログラムの事象駆動型実行モニタEDENの開発と応用, 情報処理学会論文誌, Vol. 30, No. 1, pp. 12-24 (1989).
15) German, S. M., Helmbold, D. P. and Luckman, D. C.: Monitoring for Deadlocks in Ada Tasking, *Proc. AdaTEC Conf. on Ada* (Arlington, VA), pp. 10-25 (1982).
16) Cheng, J. and Ushijima, K.: Partial Order Transparency as a Tool to Reduce Interference in Monitoring Concurrent Systems, *Distributed Environments*, Ohno, Y. (ed.), pp. 156-171, Springer-Verlag (1991).
17) Rockkind, M. J.: *Advanced UNIX Programming*, Prentice Hall International, Inc. (1985). 福崎俊博(訳)：UNIXシステムコール・プログラミング, アスキー出版局 (1991).
18) 古川善吾, 牛島和夫：並行処理プログラムのテスト法に関する一考察, 日本ソフトウェア科学会第6回大会論文集, pp. 185-188 (1989).

(平成7年4月11日受付)
(平成7年6月12日採録)



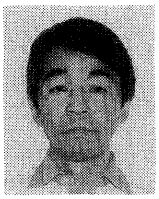
伊東 栄典（学生会員）

1969年生．1992年九州大学工学部情報工学科卒業．1994年同大学大学院情報工学専攻修了．現在，同大学院博士後期課程に在学し，ソフトウェア工学，特にソフトウェアテスト法に興味を持つ．



川口 豊（正会員）

1971年生．1993年九州大学工学部情報工学科卒業．1995年同大学大学院情報工学専攻修了．同年，沖電気工業株式会社に入社．ソフトウェア工学，特にソフトウェアテスト法に興味を持つ．



古川 善吾（正会員）

1952年生．1975年九州大学工学部情報工学科卒業．1977年同大学院情報工学専攻修士課程修了．同年日立製作所システム開発部研究所入社．1986年九州大学工学部勤務．1992年同大学情報処理教育センター助教授，現在に至る．博士(工学)．ソフトウェア工学，特にソフトウェアテスト法，日本語文書出力方式に興味を持つ．電子情報通信学会，ACM，IEEE CS各会員．

牛島 和夫（正会員）

1937年生．1961年東京大学工学部応用物理学科（数理工学コース）卒業．1963年同大学院修士課程修了．同年九州大学中央計数施設勤務．1977年九州大学工学部情報工学科教授(計算機ソフトウェア講座担当)，現在に至る．1990年4月から1994年3月まで九州大学大型計算機センター長を兼務．1991年度情報処理学会九州支部長．1995年5月から本学会監事．工学博士．電子情報通信学会，ソフトウェア科学会，ACM各会員．